DOCUMENT RESUME

ED 194 057 IR 008 907

AUTHOR Bales, W. Kenton
TITLE A Model for Generative Harmonic Dictation.
PUB DATE 80
NOTE 30p.: Paper presented at the Annual Meeting of the Association for the Development of Computer-Based Instructional Systems (Washington, DC, April 1-3, 1980).

EDRS PRICE MF01/PC02 Plus Postage.
DESCRIPTORS *Computer Assisted Instruction: *Computer Programs: Drills (Practice): *Music Theory
IDENTIFIERS *BASIC Programing Language: *Harmonic Dictation

ABSTRACT

This BASIC computer program designed to help music theory students practice harmonic dictation generates examples for students to use in a drill and practice approach in developing aural skills. To facilitate the implementation of effective generative algorithms, the author has used a non-linear analytical technique similar to the chord symbol analysis that students in a music theory classroom might use. He believes this approach is superior to the linear or Schenkerian approach used in previous research because, in order for computers to generate progressions in a linear fashion, great amounts of temporary storage must be available, and complex parallel processing techniques must be implemented in the programming. The non-linear analytical technique avoids these problems. Documentation for the computer program is included. (Author/LLS)

****
* Reproductions supplied by EDRS are the best that can be made *
* from the original document. *
****

ED194057

U.S. DEPARTMENT OF HEALTH,
EDUCATION & WELFARE
NATIONAL INSTITUTE OF
EDUCATION

THIS DOCUMENT HAS BEEN REPRODUCED EXACTLY AS RECEIVED FROM THE PERSON OR ORGANIZATION ORIGINATING IT. POINTS OF VIEW OR OPINIONS STATED DO NOT NECESSARILY REPRESENT OFFICIAL NATIONAL INSTITUTE OF EDUCATION POSITION OR POLICY

A MODEL FOR GENERATIVE HARMONIC DICTATION

by

W. Kenton Bales

"PERMISSION TO REPRODUCE THIS MATERIAL HAS BEEN GRANTED BY

W. Kenton Bales

TO THE EDUCATIONAL RESOURCES INFORMATION CENTER (ERIC)."

IR008907

GENERAL DESCRIPTION

In Figure 1, four simultaneous melodic lines combine to create a configuration which is familiar to most first semester theory students. Each line follows traditionally accepted voice leading principles common to primarily tertian tonal music from the eighteenth and early nineteenth centuries (the common practice period). Students are asked

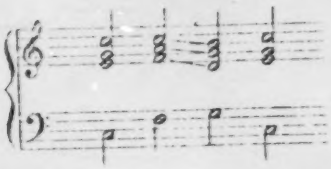


Figure 1.

to perform a varity of tasks when confronted with such an example, ranging from identifying cadential formulae to writing chord symbols and all four melodic lines. A traditional analysis of such an example might be I, IV, V, I. This analysis may be acceptable from students, but in order for a digital sound synthesis system to reproduce this example, much more information is necessary. The most common method for encoding requires the author of the lesson to enter a set of symbols for each note from a computer terminal. This method is tedious and consumes large amounts of computer memory. The number of examples possible within a given lesson as well as the size of an aural skills package may be severely limited by the amount of available magnetic storage, particularly if the system is micro- or mini-computer based. A generative method of producing aural exercises not only saves encoding time for the author, but also frees valuable computer memory so that it can be used for other purposes.

Certain consistent parameters describe chords and their progression in the common practice style. For example, three conjugations (root relationships as discussed by Eschman, 1965) describe all root motion; root position triads almost never retain the same soprano position when

the roots of the chords are a fifth apart; when the roots of two chords are a second apart, the upper three voices move in contrary motion to the bass; etc.

Progressions which are considered to be representative of the common practice style can be generated by using a circular linked list (Elson, 1975). In such a data structure, the progressions in Figure 1 appear as in Figure 2, where each of four nodes (chords) has a pointer associated with it which indicates the value of the next node (chord). Each node contains score code which describes a chord with attributes from the common practice style as described above.

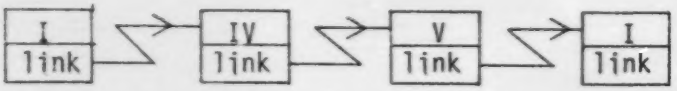


Figure 2.

Chord progressions may be generated by choosing a single symbol that is associated with score code, just as the student might respond to a given problem with at single symbol which represents an entire chordal entity. Linked list structures generated harmonic dictation examples in a manner similar to that considered by current research in linguistics (Bates, 1979).

Restraints are placed on the generator so that the limited vocabulary of the common practice period will be simulated. Part writing principles as defined by Robert Ottman in his Elementary Harmony provide bases for most restrictions imposed on the program. Triads may progress freely to one another within the confines of a tree, similar to that described by Abbot, 1978. Figure 3 represents the acceptable common chord progressions. Tonic may follow any chord and may interrupt any common chord progression. Triads may appear in root position with the root, third, or fifth in the soprano.

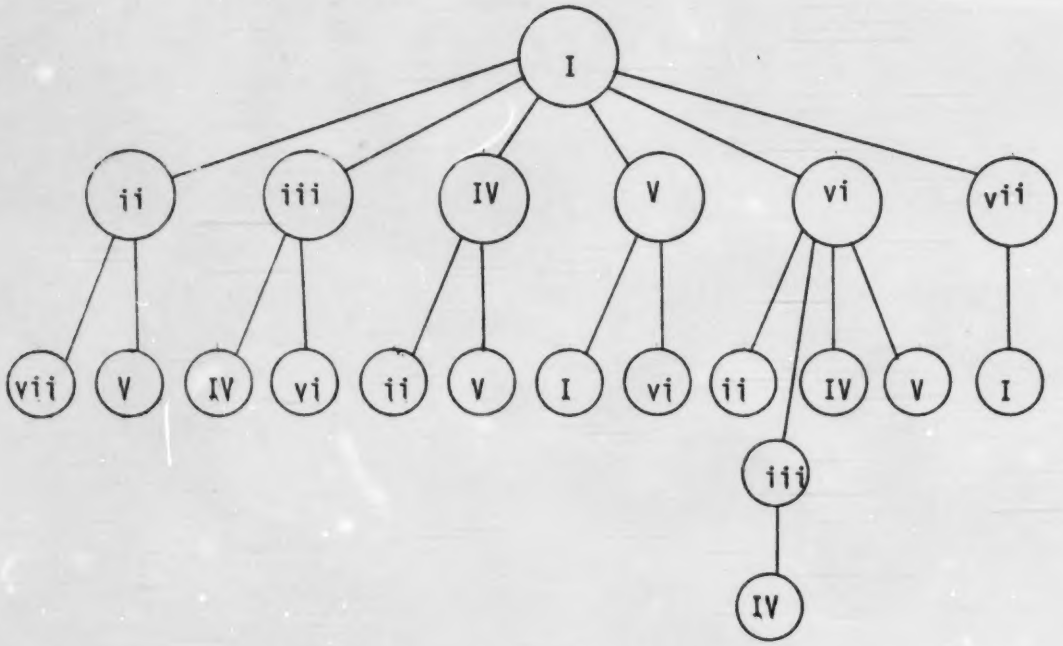


Figure 3.

Each node of the linked list must be expanded to include several pointers which allow complex branching capabilities in order to implement progressions which can be derived from Figure 3. Figure 4 is an example of how such a node might appear as a member of a file. In actual practice, many pointers are associated with each node. The program always begins at node 1, which contains score initialization information and pointers to all possible starting chords. After the first pointer is chosen,

1)	2)	3)					4)				5)
2	I	4DC	3G	2E	CUH	%E	3	5	7	8	-1

1) Number of the node
2) Correct answer string
3) Score code (for Musor)
4) Pointers to other nodes
5) End-of-node flag

Figure 4.

using a random number generator, the corresponding node is located. The program reads the correct answer and concatenates it into an answer string. The score code is read into the Musor instrument (Bales, Hamilton, and Scott, 1978) and stored until a performance is initiated. The random generator picks another number and the process is repeated until a node is reached that carries an end-of-progression flag. The score code in this node includes the play command, and the entire progression is performed for the student. His answer is checked against the concatenated answer string, and action is taken in response to his correct of incorrect entry. Default values and reatures of the AMUS language (Hamilton and Scott, 1978) make rapid score entry possible, and the transposition function allows all code in the file to stored in a single key and transposed to any other key at random.

Previous research has shown that certain chord progressions are significantly more difficult for students than are others (Hofstetter, 1978). Weighting is done in order to present more drill on problem areas and to simulate more closely common practice procedures. This weighting may be accomplished either in the driving program or in the file. (Two pointers to a single node double the possibility of that node being chosen next.) Weighting is also helpful in creating chord progressions that are of appropriate length, since the generative algorithm will produce progressions of varying length. At any time, program control may specify the length of a progression.

THE PROGRAM

The program MAIN is structured so that the code in lines 10 to 1000 monitor the overall execution of the supporting subroutines. The DIM statement in line 30 allocates space on the system for string variables and matrices. A$ is the variable that receives the user's input as an answer to the questions presented. It is dimensioned the maximum allowable size so that string length errors will not arise when the user types in excess characters. A1$ contains the correct answer as generated by the subroutine beginning in line 4000. It is also dimensioned maximum length since it can contain a varying number of characters. It will always be more than ten characters. N$ holds the name of the user. Addressing the user by name helps emphasize a feeling of individualization in instruction. Y$ acts as a repository for all yes or no answers from the user. These responses are used to facilitate branching in several places throughout the program.

The first few lines of the program (100-250) have to do with access and introduction to the system. Upon execution, the program asks for the user's name (line 100), and whether or not the user has had previous access to the program (line 120). If the answer if YES, the program asks if the user would like to review instructions on how to use the package (line 180). If the answer is NO, then the instructions are automatically presented. Y, YES, YEA, YEP, and any other response beginning with the letter Y will be accepted, since the comparison is made on the first character of the input string only. The same situation exists if the first character of the response is the letter N. A negative response is assumed and appropriate action is taken. The variable Y$ is dimensioned with a length of 10 to allow for extraneous input characters. If the response does not begin with either Y or N, a prompt is printed in order to tutor the user on appropriate responses (line 160). If the user has had previous access to the program, and does not wish to review the instructions, then the question presentation begins immediately (line 250).

As the lesson is entered, three counters which are used later in the program are initialized at zero (lines 253 and 257). K1 represents

the number of questions the user answers correctly on the first try, F1 is the flag for the repeat of the lesson, and K holds the number of tries the user has made on the current question. MAIN considers ten questions to be a lesson. The loop initialized in line 255 accounts for these ten presentations. The variable I is its index.

The first task inside the loop is the generation of the question and the correct answer. The statement GOSUB 4000 in line 270 accomplishes this by calling the subroutinedesigned specifically for this purpose. Upon return, the program clears the screen, and prints the problem number in the upper left hand corner of the screen (line 290), then waits for the user to enter an answer (line 300). If the user has typed //S, indicating a desire to terminate the lesson, the program is closed out (line 303), otherwise, the answer string is passed to the subroutine in line 7000, where it is processed and checked for accuracy. After the required computation is completed, the data may be passed to another subroutine in line 5000 for statistical calculations (line 470). When return is reached in this subroutine, the end of the question presenting loop is encountered, and the process described above is repeated (line 490).

If the lesson is terminated with the //S command, or all ten questions have been presented, the routine beginning in line 500 closes the lesson. The subroutine residing in line 6000 presents information concerning total performance on the lesson. If F1, the flag for lesson repetition is set, the user may begin the lesson again (line 520), else a sign-off message is printed (line 530), and the program waits a few seconds before it prepares itself for another user (line 540) by returning to the beginning of the program (line 550). Line 1000 is the logical end of the program, and should never be reached under normal execution procedures.

The subroutine INSTRUCT (code beginning at line 3000) is designed to present instructions to the user of the program. It is interactive and makes decisions as to how much practice is neede by the user.

The code first clears the screen by using the control 26 ('26) found in line 3010. A short general description of the purpose of the program and how the user will interact with it is presented (lines 3010 to 3040). An example is fetched from the code in line 3060 and played. [Line 3070 is a system command designed so the the program waits whil the music code is sent to Musor (Bales, Hamilton, and Scott, 1978) for translation and performance.] The program then shows the user how a correct answer would look (line 3080), and pauses long enough for the user to comprehend the material that was just presented.

When the screen is cleared again, the user is urged to try an example (line 3100). A short paragraph describes the features of the system that one needs to be familiar with for proper utilization of the program.

> "I'll play a chord progression. Listen to it and type in your answer. Be sure to type a comma between chord symbols, and hit the return key when you are finished with your answer. If you make a mistake, type the letter X while you hold down the key marked 'CTRL' on the left. This advances you to the next line where you may start over with your typing. Any time you would like me to repeat an example, just type a question mark.
>
> Here's the progression."

The most important aspects of the user's interaction include format of answer entry, use of the return key, error correction, and example repetition. Each of these points is considered in the paragraph given above. As the user becomes more acquainted with the program, other techniques will be learned that would complicate the initial contact with the system. It is important that the user does not feel intimidated by the computer.

After the sentence "Here's the progression." is printed, the score code for the example is translated and performed. Again, the system pauses during this event (line 3210). A prompt is printed that gives the user the correct answer (line 3220). At this point, the main concern is that the user knows how to respond to the system. If he or she were asked to ferret out the correct answer and enter that answer in an unfamiliar system, too much confusion would result, and and user might become alienated.

When the user enters a response, the counter K is initialized at zero. This will count the number of tries needed before a correct answer is typed. Three trials are allowed, and if no correct entry is made, it is assumed that the instructions are misunderstood. This causes their repetition (lines 3320 and 3330). On the other hand, if either the first, second or third try is correct, the user is assumed ready to begin the actual lesson, and a return to the main program is generated (lines 3340 to 3350). At the beginning of any session, the user may review the instructions by responding affirmatively to the pertinent question presented in the main program.

GENER, the subroutine beginning at line 4000, actually produces the progressions that the user hears, and constructs the string which contains the correct answer. This routine has its own DIM statements (lines 4010 and 4020) since it requires string variables and a matrix that are not utilized by the main program. S$ is a varying length character string wnich contains AMUS score code (Hamilton and Scott, 1977) for the current node. T$ receives a character string which gives information on the transposition of the example in question (See %T below). T1$ is the result of the conversion of the random number X1, and becomes the final characters of T$. A0$ is a substring of A1$ which contains the portion of the complete answer generated by the current node. The matrix A is used to retain the list of pointers associated with each node.

The FILES statement in line 4030 identified the file H1 upon entry into the subroutine. H1 contains nodes capable of producing chord progressions using tonic, subdominant, and dominant harmonies and is described in full below. Lines 4040-4060 initialize some of the variables used by the program. The matrix A is initialized to zero (4040); P, the value of the pointer to the next node for processing, is set to 1 (4050): and A1$ is initialized to contain one blank.

The score code for the current example is initialized in the subroutine opening at line 4800, described below. The opening PRINT statement (line 4090) displays %S (indicating the beginning of a musical score), and T$, which will use the %T command to indicate the interval of transposition, and a separating blank. The example is now initialized, and the correct node is located which contains the desired information (GOSUB 4500, line 4110).

After the variables are located by the subroutine, the answer string is contructed by concatenating A1$ and A0$. This is done in line 4140 by the statement A1$(LEN(A1$),LEN(A0$)+LEN(A1$))=A0$. Or, stated verbally, the substring of A1$ beginning at the current end of A1$ and continuing for the current length of A0$, is replaced with the current contents of A0$. For example, if A1$="I,", and A0$="IV,", the resultant string will be "I,IV,".

When this calculation is completed, the score code is printed followed by a padding blank (line 4160). This blank is a syntactical necessity of the AMUS system.

The loop beginning in line 4170 loads the matrix A with values from the list of pointers associated with this node. One hundred pointers are possible, but most are not used. A pointer value is read from the file and stored in a dummy variable, Z (line 4180). If this value is an end of file flag, zero, then the example is complete and a branch to the conclusion of the routine is generated. If the value of Z is -1, line 4200, then the end of the current node has been reached and a random number is generated that will be used to calculate the value of the pointer to the next node (line 4230). A check to

assure that this number is valid is made (line 4240), and the value is assigned to P, the number of the next node for which to search (line 4250). If Z is not one of these flags, the value is loaded into the proper location in the matrix A(line 4210), and the next value is processed (line 4220).

Finally, the routin sends itself back to the top of the loop to process the next node in the list (line 4260). If the condition in line 4190 has been met (if Z ≠ 0), then the score is played by printing the %P command, and ended by displaying %E. No score code is printed on the CRT because of the particular monitoring program in the M6800 microprocessor that is part of the AMUS system.

The subroutine which begins in line 4500 locates the proper record or node of the linked list stored in the file H1. The file pointer is initialized and set at the first item (line 4510). Each node in the file has three initial variables: N, the number of the node; A0$, the answer string associated with this node; and S$, the score code necessary to produce the particular chord used in this node. These items are read in line 4520. A list of pointers follows that may be from one to one hundred numbers in length. These are read in the loop beginning in line 4540. If one of them is a flag, 0 or -1, then the next file entry will be the beginning of another node. The process is then repeated, as program control is sent to line 4520.

The crucial test is made in line 4530. P is the value of the pointer passed from the routine GENER. N is the number of the node currently being processed by this subroutine. When P and N are equal, the desired node has been located, and this information is returned to the calling procedure (line 4580). Note that the first node will always be processed before any others, since P is set to 1 in line 4050. P must never (and will not) equal zero. The latter is ensured by the scaling of the random generator found in line 4230. (I2 can never be zero, since it is loop index declared 1 to 100).

The transposition subroutine takes advantage of the %T command in the AMUS score language. This conveniently allows everything in the file to be written in only one key. The form of the command is

"%Tn", where n is an integer greater than -7 and less than 7. The first two characters are always "%T", and those are initialized in line 4810. Two consecutive random number generators, X and X1, provide integers for the latter part of the command. Line 4280 generates a number between zero and seven. Based on the condition of X1, which is either on or off (zero or one), X is multiplied by minus one so that scaling in the negative range may occur (lines 4830-4850). This integer value is converted into a character string named T1$ (line 4860), and concatenated with the pre-existent T$ (line 4870). Return to the calling routine ensues (line 4880).

CLOSE is a short subroutine that is accessed after all questions in a given lesson are presented, or the user terminates the program with the //S command.

The screen is cleared, and a termination message is printed (line 6010). Information concerning the user's accuracy is displayed in the following format:

```
You got K1 out of I-1 correct.
That's (K1/(I-1))*100 percent.
```

K1 is the number of questions answered correctly, and I-1 is the total number of questions presented. The percentage calculation in line 6025 is scaled by 100 so that the information appears on the left-hand side of the decimal point.

If the user would like to repeat the lesson, and affirmative answer to the question posed in line 6030 sends the program back to the beginning. A negative answer here causes a farewell message to be produced (line 6090), and the system pauses while the user has time to move away from the terminal (line 6100). The flag F1 is the signal for the repeat of each lesson, and is reset at the beginning of each lesson. After the system pause, the program begins by asking for the name of the next user.

This closing routine puts the program in an endless loop that may be stopped only by depression of the BREAK key. Such design allows for maximum run-time efficiency when several users are on the system simultaneously, and frees the laboratory monitor or other personnel from re-executing the program between users.

The design of the file utilized by the program MAIN is a primary consideration. It is constructed so that alterations to the file create changes in the lesson, rather than necessitating changes in the program itself. Any type of presentation that can be described by a linked list may be implemented by careful construction of a file for use by this program.

In musical terms, this design affords itself easily to the generation of harmonic dictation examples. The file described here creates progressions containing only the tonic, sub-dominant, and dominant harmonies, since most problems in part-writing can be demonstrated using only these three chords.

Several limitations are imposed on the relationships between these chords. First, three voicings of any given chord are the only ones allowed. Although many more are possible, the expansion necessary in the file to accomadate the increased size is prohibitive, and the logic involved in linking one chord to another becomes disjointed and more complicated than is generally considered necessary. This allows for all three soprano positions and provides more than enough variety for most undergraduate theory students.

A particular combination of open and closed position is necessary in order to ensure proper voice leading. For instance, a triad voiced in closed position with the root in the soprano may not move smoothly to another triad voiced similarly (consecutive octaves and fifths rear their ugly heads here), but if the position of the second chord is changed, proper part-writing is assured and the student has a "correct" example foraural analysis. Unusual enharmonic spellings of triads appear throughout the file in order to avoid the problems encountered in octave register shifts in the AMUS language. By using b sharps instead of c's, the second voice is allowed to stay in the same octave for the entire progression. C flats permit the soprano voice to remain in one relative octave.

The primary node of the file carries score initialization information, and pointers to any of three starting places for the progression. Each progressions starts with the tonic triad, but any of three soprano positions are possible, as discussed above.

All consecutive nodes contain several entries, in alpha-numeric and numeric code. Each node begins with a number which represents the number of the node in the file. The next entry is a string which is the portion of the correct answer for this node of the file. It is always an upper case Roman numeral followed by a comma, unless the node is one tof the final entries. These entries have no commas after them and indicate the end of the score. Following the answer string, a string of AMUS score code expressing the triad for this node is encountered. All entries are in the key of C major and are transposed to other keys through the use of the %T command. A list of pointers follows, and the final entry is the end-of-node (-1), or the end-of-example flag (0). Three nodes may close the example in the file under discussion, and each are tonic triads.

The progressions are weighted by increasing the number of pointers to the desired node. For example, two pointers with the value 7, and one pointer with the value 5 would increase the likelihood of the seventh node being chosen next.

APPENDIX A

Program listings

```
10  REM * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
11  REM *                                                                     *
12  REM *      THIS PROGRAM IS A GENERATOR AND DRIVER FOR A GENERATIVE         *
13  REM *    HARMONIC DICTATION DRILL AND PRACTICE LESSON.  IT MAY BE         *
14  REM *    USED FOR ANY NUMBER OF LESSON TYPES.                             *
15  REM *                                                                     *
16  REM *                                                                     *
17  REM *                  PROGRAM WRITTEN BY KEN BALES                       *
18  REM *                              12/79                                  *
19  REM *                                                                     *
20  REM * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
25  REM
30  DIM A$[255],A1$[255],C$[8],N$[20],Y$[3]
100  PRINT '26"WHAT IS YOUR NAME? (FIRST NAME ONLY, PLEASE.)";
110  LINPUT N$
120  PRINT LIN(2);TAB(10);"HAVE YOU EVER USED THIS PROGRAM BEFORE?";
130  LINPUT Y$
140  IF Y$[1,1]="Y" THEN 180
150  IF Y$[1,1]="N" THEN 240
160  PRINT "TYPE 'Y' OR 'N' ONLY FOR YES OR NO."
170  GOTO 130
180  PRINT LIN(4);TAB(5);"WOULD YOU LIKE INSTRUCTIONS?";
190  LINPUT Y$
200  IF Y$[1,1]="Y" THEN 240
210  IF Y$[1,1]="N" THEN 250
230  GOTO 190
240  REM
241  REM   GO GET INSTRUCTIONS
242  REM
245  GOSUB 3000
250  REM
251  REM     BEGIN QUESTION PRESENTATION
252  REM
253  K=K1=0
255  FOR I=1 TO 10
260  REM
261  REM     GENERATE THE QUESTION
262  REM
270  GOSUB 4000
280  REM
281  REM     PRESENT THE QUESTION TO THE USER
282  REM
285  K=K+1
290  PRINT '26"NUMBER ";I;
295  REM
296  REM     USER ENTERS THE ANSWER
297  REM
300  LINPUT A$
303  GOSUB 7000
305  IF A$[1,3]="//S" THEN 500
310  IF A$=A1$ THEN 340
315  IF K>1 THEN 456
320  PRINT "NO, LISTEN AND TRY AGAIN";
330  GOTO 295
340  REM
341  REM     ANSWER IS CORRECT
```

```
342  REM
345  K1=K1+1
350  X=INT(RND(TIM(4))*5)+1
360  GOTO X OF 370,390,410,430,450
370  PRINT "RIGHT, "+N$+"!"
380  GOTO 460
390  PRINT "THAT'S CORRECT."
400  GOTO 460
410  PRINT "GOOD, "+N$+"!"
420  GOTO 460
430  PRINT "YOU'VE GOT IT!"
440  GOTO 460
450  PRINT "WELL DONE, "+N$+"."
455  GOTO 460
456  REM  GIVE THE CORRECT ANSWER AFTER 2 MISSES.
457  PRINT LIN(2)+"SORRY, THE CORRECT ANSWER IS "+A1$+"."
458  SYSTEM R3,"PAUSE-5"
460  REM
461  REM      PROCESS THIS QUESTION
462  REM
470  GOSUB 5000
480  REM
481  REM      DO NEXT QUESTION
482  REM
490  NEXT I
500  REM
501  REM      CLOSE OUT THIS LESSON
502  REM
510  GOSUB 6000
520  IF F1 THEN 250
530  PRINT "BYE-BYE!"
540  SYSTEM R3,"PAUSE-30"
550  GOTO 100
1000  STOP
3000  REM
3001  REM - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
3002  REM
3003  REM      THIS SUBROUTINE GIVES THE USER INSTRUCTIONS FOR THE PROGRAM
3004  REM
3005  REM - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
3006  REM
3010  PRINT '26LIN(2)+"THIS PROGRAM IS DESIGNED TO HELP YOU PRACTICE
      HARMONIC DICTATION."
3020  PRINT LIN(2)+"WHEN I PLAY A PROGRESSION, TYPE IN THE ROMAN
      NUMERALS WHICH"
3030  PRINT TAB(3)+"REPRESENT IT.  TYPE UPPER CASE FOR MAJOR, AND"
3040  PRINT TAB(5)+"LOWER CASE FOR MINOR."
3050  PRINT LIN(3)+"LISTEN TO AN EXAMPLE:"
3060  PRINT "%S 4CD 3GD 2E- 1CUH 4G 3G 2D 1BDH 4C 3G 2E- 1CUH %P %E"
3070  SYSTEM R3,"PAUSE-15"
3080  PRINT LIN(4)+"YOU WOULD HAVE TYPED 'i,V,i'."
3090  SYSTEM R3,"PAUSE-10"
3100  PRINT '26"LET'S TRY AN EXAMPLE, "+N$+"."
```

```
3110  PRINT LIN(4);"I'LL PLAY A CHORD PROGRESSION.  LISTEN TO IT AND TYPE IN"
3120  PRINT TAB(3);"YOUR ANSWER.  BE SURE TO TYPE A COMMA BETWEEN CHORD SYMBOLS"
3130  PRINT TAB(5);"AND HIT THE RETURN KEY WHEN YOU ARE FINISHED TYPING."
3140  PRINT LIN(2);"IF YOU MAKE A MISTAKE, TYPE 'X' WHILE YOU HOLD DOWN THE "
3150  PRINT TAB(3);"KEY MARKED 'CTRL' ON THE LEFT.  THIS ADVANCES YOU TO THE"
3160  PRINT TAB(5);"NEXT LINE SO THAT YOU MAY START OVER WITH YOUR TYPING."
3170  PRINT LIN(2);"ANYTIME YOU WOULD LIKE ME TO REPEAT AN EXAMPLE, JUST"

3180  PRINT TAB(3);"TYPE A QUESTION MARK."
3190  PRINT LIN(2);"HERE'S THE PROGRESSION."
3195  PRINT "%S 4DD 3AD 2F 1DUH 4AD 3A 2E 1C$H 4D 3A 2F 1DH %P %E"
3210  SYSTEM R3,"PAUSE-15"
3220  PRINT LIN(2);"NOW, TYPE IN YOUR ANSWER (i,V,i)."
3230  K=0
3240  LINPUT A$
3250  IF K>2 THEN 3310
3270  IF A$="i,V,i" THEN 3340
3280  K=K+1
3290  PRINT "THAT'S NOT CORRECT, TRY AGAIN."
3300  GOTO 3240
3310  PRINT "YOU SHOULD HAVE TYPED 'i,V,i'."
3320  PRINT "LET'S LOOK AT THE INSTRUCTIONS AGAIN."
3330  GOTO 3020
3340  PRINT LIN(2);"CORRECT, ";N$;". LET'S GET STARTED."
3350  RETURN
```

```
4000  REM  THIS SUBROUTINE GENERATES THE SCORE CODE FOR THE EXAMPLE.
4010  DIM S$[255],A[100],T$[4],X1$[2]
4020  DIM A0$[3],A1$[255]
4030  FILES H1
4040  MAT A=ZER
4050  P=1
4060  A1$=" "
4070  REM INITIALIZE SCORE CODE
4080  GOSUB 4800
4090  PRINT "%S %W ";T$;" ";
4100  REM  TOP OF LOOP
4110  GOSUB 4500
4120  PRINT A0$
4130  REM CONSTRUCT ANSWER STRING
4140  A1$[LEN(A1$),LEN(A0$)+LEN(A1$)]=A0$
4150  PRINT "A1$=";A1$
4160  PRINT S$;" ";
4170  FOR I2=1 TO 100
4180  READ #1;Z
4190  IF Z=0 THEN 4900
4200  IF Z=-1 THEN 4230
4210  A[I2]=Z
4220  NEXT I2
4230  X=INT(RND(TIM(4))*I2)+1
4240  IF X<1 OR X >= I2 THEN 4230
4250  P=A[X]
4260  GOTO 4100
4500  REM
4502  REM      THIS SUBROUTINE LOCATES THE PROPER RECORD FOR THE CHORD
4505  REM
4510  READ #1,1
4520  READ #1;N,A0$,S$
4530  IF P=N THEN 4580
4540  FOR I3=1 TO 100
4550  READ #1;N
4560  IF N=0 OR N=-1 THEN 4520
4570  NEXT I3
4580  RETURN
4800  REM
4802  REM      THIS SUBROUTINE CHOOSES THE TRANSPOSITION FOR THIS
                PROGRESSION
4805  REM
4810  T$[1,2]="%T"
4820  X=INT(RND(TIM(4))*6)+1
4830  X1=INT(RND(TIM(4))*2)
4840  IF X1 THEN 4860
4850  X1=X1*-1
4860  CONVERT X1 TO T1$
4870  T$[3,LEN(T1$)+3]=T1$
4880  RETURN
4900  PRINT "%P %E"
4999  RETURN
```

```
5000  REM
5001  REM - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
5002  REM
5003  REM      THIS SUBROUTINE WILL PROCESS STATISTICS ON THE LESSON
5004  REM
5005  REM - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
5006  REM
5999  RETURN
6000  REM
6001  REM - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
6002  REM
6003  REM      SUBROUTINE TO CLOSE OUT THE LESSON
6004  REM
6005  REM - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
6006  REM
6010  PRINT '26"YOU HAVE FINISHED THIS LESSON, "$N$$"."
6020  PRINT LIN(2)$"YOU GOT"$K1$"OUT OF"$I-1$"CORRECT FOR"$
6025  PRINT (K1/(I-1))*100$"PERCENT."
6030  PRINT LIN(2)$"WOULD YOU LIKE TO DO MORE?"
6040  LINPUT Y$
6050  IF Y$[1,1]="Y" THEN 6080
6060  F1=0
6070  GOTO 6999
6080  F1=1
6999  RETURN
7000  REM
7001  REM - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
7002  REM
7003  REM      THIS SUBROUTINE WILL PROCESS THE USER'S INPUT
7004  REM
7005  REM - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
7006  REM

9999  END
```

```
10  DIM A$[3],S$[14]
20  FILES H1
30  REM PROGRAM TO LOAD FILES
35  PRINT " ENTER THE NUMBER ZERO WHEN ALL NODES ARE COMPLETED.";LIN(3)
40  PRINT "ENTER NODE NUMBER:"
50  INPUT N
60  IF N<1 THEN 190
70  PRINT #1;N
80  PRINT "ENTER ANSWER:"
90  LINPUT A$
100  PRINT #1;A$
110  PRINT "ENTER SCORE CODE:"
120  LINPUT S$
130  PRINT #1;S$
140  PRINT "ENTER POINTERS:"
150  INPUT P
160  PRINT #1;P
170  IF P<1 THEN 40
180  GOTO 150
190  PRINT "ALL DONE. BYE-BYE."
999  END
```

This program was used to load the nodes in the file.

APPENDIX B

File Listings

```
# 1

4%03 1%05
# 2
# 3
# 4
#-1


# 2
I,
4C 3E 2G CH
# 3
# 4
# 5
# 5
# 5
# 8
# 8
# 8
# 9
# 9
# 9
#-1


# 3
I,
4C 3C 2G EH
# 2
# 4
# 5
# 5
# 6
# 6
# 8
# 8
# 9
# 9
# 9
#-1


# 4
I,
4C 3E 2B# GH
# 2
# 3
# 6
# 6
# 6
# 7
# 7
# 7
# 8
# 8
# 10
# 10
# 10
#-1


# 5
IV,
4F 3F 2A CH
# 6
# 7
# 2
# 2
# 8
# 8
# 8
#-1


# 6
IV,
4F 3C 2A FH
# 5
# 7
# 8
# 8
# 9
# 9
# 9
#-1
```

```
# 7
IV,
4F 3F 2B# AH
# 5
# 6
# 10
# 10
# 10
#-1
```

```
# 8
V,
4B 3D 2B C-H
# 11
# 2
# 11
# 9
# 11
# 10
# 11
# 11
#-1
```

```
# 9
V,
4G 3G 2B DH
# 11
# 8
# 12
# 11
# 10
# 12
# 11
# 12
# 2
# 3
#-1
```

```
# 10
V,
4G 3D 2B GH
# 8
# 9
# 13
# 4
# 13
# 4
# 13
# 3
# 13
# 3
#-1
```

```
# 11
I
4C 3E 2B CH
# 0
# 12
I
4C 3C 2B EH
# 0
# 13
I
4C 3E 2B# GH
# 0
```

VARIABLE LIST

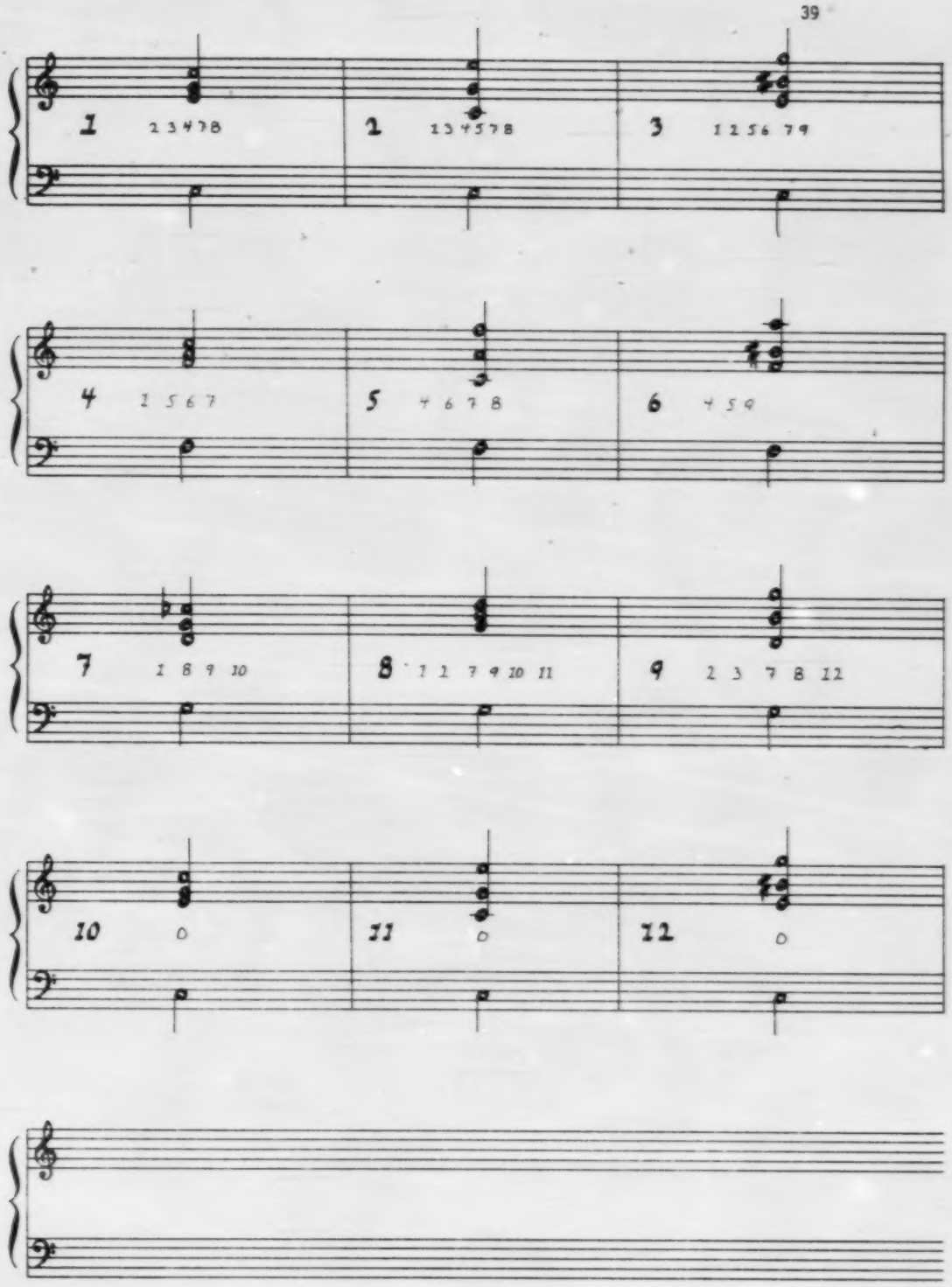
CHARACTER STRINGS

A$ - String containing user's response to question
AO$ - substring of correct answer in each node
A1$ - string containing correct answer
N$ - string containing user's name
S$ - score code for individual chords
T$ - transposition string in AMUS code
T1$ - part of T$ that contains transposition number
Y$ - yes or no response string

MATRICES

A- matrix of pointers associated with each node

NUMERIC VARIABLES

F1 - flag for repeat of lesson
I - loop to generate questions throughout each lesson
I1 - loop index
I2 - loop index for matrix loading routine
I3 - loop to read past file entries until desired node is found
K - counter for number of tries on current question
K1 - counter for number of answers correct on first try
N - number of node being processed
P - set to compare for pointer to next node
R3 - return variable for system pause
X - result of random number generator
X1 - result of random number generator
Z - dummy variable to read through matrix of pointers

Alpheus Music Corp
Hollywood, Calif.
M-107

REFERENCES

Abbot, Curtis. "Machine Tongues III". Computer Music Journal, Vol. II, No. 3, 1978.

Bales, W. Kenton, Richard L. Hamilton and Dan W. Scott. Computerized composition and performance with AMUS. Proceedings of the 1978 International Computer Music Conference. Evanston, Illinois, 1978, 450-470.

Bates, Madeleine and Kirk Wilson. A generative computer system to teach language to the deaf. Proceedings of the Annual Convention of the Association for the Development of Computer-Based Instructional Systems. San Diego, California, 1979, 413-424.

Elson, Mark. Data Structures. Boston, Massachusetts: Science Research Associates, Inc., 1975.

Eschman, Karl. Teaching Music Theory. Boston, Massachusetts: E. C. Schirmer Music Company, 1965.

Frankel, Robert E. The modeling of musical perception on a digital computer via Schenkerian theory. Paper presented at the Annual Meeting of the American Musicological Society. Los Angeles, California, 1975.

Hamilton, Richard L. and Dan W. Scott. A new approach to computer assisted instruction in music theory. Proceedings of the Ninth Conference on Computers in the Undergraduate Curriculum. Denver, Colorado, 1978.

Hofstetter, Fred T. Computer-based recognition of perceptual patterns in harmonic dictation exercises. Proceedings of the Annual Convention of the Association for the Development of Computer-Based Instructional Systems. Dallas, Texas, 1978, 192-201.

Keiler, Allan R. "The Syntax of Prolongation I (Within Context of Theories of Heinrich Schenker)." In Theory Only, Vol. III, No. 5, 1977.

Licklider, J. C. R. "Impact of Information Technology on Education in Science and Technology." Technology in Science Education: The Next Ten Years, prepared for the National Science Foundation, July, 1979, 1-10.

Ottman, Robert W. Elementary Harmony, 2nd ed. Englewood Cliffs, New Jersey: Printice-Hall, Inc., 1970.